# Tratamento de erros I

- ***Ideia básica***
  - Sempre que ocorre um erro, o seu tratamento fica a cargo da última declaração '**`On Error`**' invocada no procedimento corrente
  - Caso não haja qualquer declaração desse tipo, o erro é tratado no primeiro procedimento pai que possui tratamento de erros. Se nenhum dos procedimentos envolvidos possuir tratamento de erros, o sistema aborta a execução e apresenta uma mensagem indicando o erro ocorrido
- ***Formas de tratar erros***
  - **`On Error GoTo line`**: activa a rotina de tratamento de erros que se encontra a partir da indicação **`line`**. A posterior ocorrência de qualquer erro transfere imediatamente a execução para a rotina especificada

```
[Sub | Function] procedimento( )
  On Error GoTo Rotina_Tratamento_Erros
    [...]
    Exit [Sub | Function]
  Rotina_Tratamento_Erros:
    [...]
End [Sub | Function]
```

# Tratamento de erros II

- ***Formas de tratar erros***
  - **`On Error Resume Next`**: ignora a posterior ocorrência de qualquer erro e prossegue a execução na instrução seguinte à que provocar o erro
  - **`On Error GoTo 0`**: desactiva o tratamento de erros no procedimento corrente
- ***Retomar a execução***
  - **`Resume`**: retoma a execução na instrução que provocou o erro
  - **`Resume Next`**: retoma a execução na instrução seguinte à que provocou o erro
  - **`Resume line`**: retoma a execução na instrução indicada por **`line`**
- ***Objecto Err***
  - **`Err.Number`**: número que identifica o último erro ocorrido
  - **`Err.Description`**: descrição sumária do último erro ocorrido
  - **`Err.Clear`**: limpa as propriedades **`Err.Number`** (0) e **`Err.Description`** (“”). É igualmente invocado sempre que uma das seguintes instruções é executada:
    - Instruções do tipo **`Resume`**
    - Instruções do tipo **`On Error`**
    - Instruções **`Exit Sub`**, **`Exit Function`**, **`Exit Property`**
  - **`Err.Raise Number`**: provoca a ocorrência do erro identificado por **`Number`**

# Tratamento de erros III

- *Exemplo*

```
Sub teste_erros()
  On Error GoTo trat_erros:
  Dim erro As Integer : erro = 5 : MsgBox "Início teste_erros"
  gera_erro (erro)
  MsgBox "Fim teste_erros" : Exit Sub
trat_erros:
  MsgBox "Erro n. " & Err.Number
  Select Case Err.Number
    Case 5 To 6
      erro = erro + 1 : Resume
    Case Else
      Resume Next
  End Select
End Sub

Sub gera_erro(erro As Integer)
  MsgBox "Início gera_erro"
  Err.Raise erro
  MsgBox "Fim gera_erro"
End Sub
```

- *Sequência de mensagens*
  - Início teste_erros
  - Início gera_erro
  - Erro n. 5
  - Início gera_erro
  - Erro n. 6
  - Início gera_erro
  - Erro n. 7
  - Fim teste_erros

# Collections

- ***Objecto Collection***
  - Conjunto ordenado de objectos não necessariamente do mesmo tipo
  - Declarar um novo objecto: `Dim col As New Collection`

- ***Propriedades e métodos***
  - **`collection.Count`**: número total de objectos na *collection* (só de leitura)
  - **`collection.Add Item, Key, Before, After`**: adiciona um novo objecto
    - `Item`: objecto a adicionar
    - `Key`: nome pelo qual o objecto será conhecido na *collection*
    - `Before`/`After`: objecto antes/depois do qual o novo objecto deve ser adicionado
  - **`collection(Index)`**: objecto de ordem **`Index`** (o primeiro tem ordem 1)
  - **`collection(Key)`**: objecto cujo nome é **`Key`**
  - **`collection.Remove Index`**: remove o objecto de ordem **`Index`**
  - **`collection.Remove Key`**: remove o objecto cujo nome é **`Key`**

# Módulos de classe I

- *Exemplo*

```
‘para trabalhar com facturas posso declarar um conjunto de
‘variáveis que represente os dados que pretendo manipular
Dim ID As Long  :  Dim dataFactura As Date
Dim dataVencimento As Date  :  Dim dataPagamento As Date
Dim quantia As Currency  :  Dim emAtraso As boolean

‘para trabalhar com diferentes facturas em simultâneo posso
‘criar um novo tipo de dados que represente a factura
Type Factura
  Dim ID As Long  : ... :  Dim emAtraso As boolean
End Type
Dim factura01 As Factura : Dim factura02 As Factura

‘como a manipulação de facturas envolve quase sempre as
‘mesmas operações, posso criar um módulo de classe que
‘especifique um novo objecto do tipo factura com propriedades
‘e métodos que manipulem as facturas de igual modo
Dim factura01 As New CFactura : Dim factura02 As New CFactura
```

# Módulos de classe II

- *Variáveis de classe (data members)*
  - `Private m_DataMember As Type`
- *Propriedades de classe (Property Get / Property Let)*

```
Public
Property Get PropertyName([args]) As PropertyDataType
  [... : PropertyName = expr]
  [Exit Property]
  [... : PropertyName = expr]
End Property

Public
Property Let PropertyName([args,] NewVal As PropertyDataType)
  [... : m_DataMember = expr]
  [Exit Property]
  [... : m_DataMember = expr]
End Property
```

# Módulos de classe III

- *Exemplo*

```
'data members do módulo de classe CFactura
Private m_ID As Long  : ... :  Private m_emAtraso As boolean
'factura01.ID
Public Property Get ID() As Long
    ID = m_ID
End Property
'factura01.EmAtraso
Public Property Get EmAtraso() As Boolean
    EmAtraso = m_emAtraso
End Property
'factura01.Quantia = newQuantia
Public Property Let Quantia(newQuantia As Currency)
  If newQuantia < 0 Then
    newQuantia = 0
    MsgBox "Quantia inválida! Será usado zero."
  End If
  m_quantia = newQuantia
End Property
```

# Módulos de classe IV

- *Exemplo*

```
'factura01.DataFactura = newData
Public Property Let DataFactura(newData As Date)
  If newData > Date Then
    newData = Date
    MsgBox “Data inválida! Será usada a data de hoje.”
  End If
  m_dataFactura = newData
  m_dataVencimento = m_dataFactura + 30
End Property

'factura01.DataPagamento = newData
Public Property Let DataPagamento(newData As Date)
  m_dataPagamento = newData
  m_emAtraso = False
End Property
```

# Módulos de classe V

- *Propriedades de classe (Property Set)*

```
Public
Property Set PropertyName([args,] NewObj As PropertyDataType)
  [... : Set m_DataMember = expr]
  [Exit Property]
  [... : Set m_DataMember = expr]
End Property
```

- *Métodos de classe*

```
Public [Sub|Function] MethodName([args]) [As MethodDataType]
  [...]
End [Sub|Function]
```

- *Exemplo*

```
Public Sub Update()                          'factura01.Update
  If m_dataPagamento = 0 And m_dataVencimento < Date Then
    m_emAtraso = True
  End If
End Sub
```

# Módulos de classe VI

- ***Eventos de classe***
  - **`Private Sub Class_Initialize( )`**: ocorre quando um objecto da classe é instanciado pela primeira vez
  - **`Private Sub Class_Terminate( )`**: ocorre quando todos os objectos da classe são removidos
- ***Exemplo***

```
Private Sub Class_Initialize()
  Randomize
  m_ID = Int(Rnd * 999999)
  m_dataFactura = Date
  m_dataVencimento = m_dataFactura + 30
  m_emAtraso = True
End Sub

Private Sub Class_Terminate()
  ...
End Sub
```

# Módulos de classe VII

- *Classes de collections*
  - `Private m_DataMember As New Collection`
- *Exemplo*

```
'para manipular múltiplas facturas pode ser útil criar um
'novo módulo de classe que especifique um objecto facturas
Dim facts As New CFacturas

'data member do módulo de classe CFacturas
Private m_facturas As New Collection

'facts.Add quantia
Public Function Add(quantia As Currency) As CFactura
  Dim new_fact As New CFactura
  With new_fact
    .Quantia = quantia                'Property Let de CFactura
    m_facturas.Add Item:=new_fact, Key:=CStr(.ID)
  End With
  Set Add = new_fact
End Function
```

# Módulos de classe VIII

- *Exemplo*

```
'facts.Remove index
Public Sub Remove(index As Variant) 'index é do tipo Variant
  m_facturas.Remove index           'porque pode ser um
End Sub                             'número ou uma string
'facts.Count
Public Function Count() As Long
  Count = m_facturas.Count
End Function
'facts.Item index
Public Function Item(index As Variant) As CFactura
    Set Item = m_facturas(index)
End Function
'exemplo de utilização
Dim facts As New CFacturas
facts.Add 1000
facts.Add 2000
MsgBox facts.Count                  'apresenta "2"
facts.Remove 1
MsgBox facts.Item(1).Quantia        'apresenta "2000"
```

# Módulos de classe IX

- ***Propriedades de classe como objectos***
  - Muitas das propriedades de um objecto são por si só também objectos. Como é então possível definir uma propriedade como objecto?
- ***Exemplo***

```
'suponhamos que queremos criar um módulo de classe CCliente
'que especifique um novo objecto cliente
Private m_Nome As String
Private m_Facturas As CFacturas

'cliente01.Nome
Public Property Get Nome() As String
    Nome = m_Nome
End Property

'cliente01.Facturas
Public Property Get Facturas() As CFacturas
  Set Facturas = m_Facturas
End Property
```

# Módulos de classe X

- *Exemplo*

```
'cliente01.Nome = newNome
Public Property Let Nome(newNome As String)
    m_Nome = newNome
End Property
'Set cliente01.Facturas = newFacturas
Public Property Set Facturas(newFacturas As CFacturas)
  Set m_Facturas = newFacturas
End Property
'exemplo de utilização
Dim cliente01 As New CCliente
With cliente01
  .Nome = "António"
  Set .Facturas = New CFacturas  'a declaração New cria uma
  With .Facturas                 'nova instância do objecto
    .Add 1000
    .Add 2000
  End With
End With
MsgBox cliente01.Facturas.Item(1).Quantia  'apresenta "1000"
```